VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2016

Martin Mohler

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Mohler**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: INOvativní služby V IT, s.r.o
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Michal Krátký, Ph.D.**

Konzultant bakalářské práce: Michal Štorkán

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.

V Ostravě 29. 4. 2016 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. 4. 2016 .....

Chtěl bych poděkovat skvělému kolektivu ze společnosti INOvativní služby V IT s.r.o. za možnost získání spousty nových znalostí a zkušeností. Zejména pak možnosti se seznámit i s nejnovějšími technologiemi. Jmenovitě bych rád poděkoval Michalu Štorkánovi za vedení mé praxe ve firmě a Rostislavu Urbánkovi za skvělé vedení a předání cenných znalostí.

Abstrakt

Tato bakalářská práce popisuje průběh odborné praxe ve firmě INOvativní služby V IT s.r.o. V první části bakalářská práce popisuje zaměření společnosti a pracovní zařazení studenta. Druhá část obsahuje výpis úkolů vykonané v rámci praxe ve firmě. Popis řešení těchto úkolů je ve třetí části této bakalářské práce za pomocí PHP, frameworku Nette a Google Cloud služeb. Závěrem práce je shrnutí znalostí využitých při plnění úkolů a celkové zhodnocení této praxe.

Klíčová slova: Bakalářská práce, odborná praxe, INOvativní služby V IT s.r.o., PHP, Framework Nette, App Engine, Compute Engine, Cloud SQL, Google Bucket, HTML, CSS, JavaScript, XML

Abstract

The thesis describes the course of professional practice in company called INOvativní služby V IT s.r.o. The first part describes the company fucus and the grade of student. The second part contains a list of tasks carried out during this practice in the company. The third part describes all the solutions of these tasks with PHP, framework Nette and Google Cloud service. Finally, there is a summary of knowledge used to complete the tasks and overall assessment of this practice.

Keywords: Thesis, professional experience, INOvativní služby V IT s.r.o., PHP, Nette Framework, App Engine, Compute Engine, Cloud SQL, Google Bucket, HTML, CSS, JavaScript, XML

Obsah

Seznam použitých zkratek a symbolů

API	-	Application Programming Interface
CK	-	Cestovní Kancelář
CSS	-	Cascading Style Sheets
CSV	-	Comma-Separated Values
HTML	-	HyperText Markup Language
IT	-	Informační Technologie
JS	-	JavaScript
MVP	-	Model – View - Presener
PHP	-	Hypertext Preprocessor
RAM	-	Random Access Memory
XML	-	Extensible Markup Language

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce je popisem mé praxe ve firmě INOvativní služby V IT s.r.o. sídlící v Ostravě. K praxi ve firmě INOvativní služby V IT s.r.o. jsem se dostal náhodou, když jsem přes facebookové stránky narazil na inzerát s nabídkou pozice HTML/CSS kodér - junior. Po přijímacím řízení jsem se domluvil na vykonávání praxe, protože firma i její projekty mě velmi zaujaly.

Společnost INOvativní služby V IT s.r.o. mi umožnila rozšířit mé znalosti nejen kódovacích jazyků jako je HTML, CSS, ale také programovacích jazyků typu JavaScript a hlavně PHP. Díky mému rozvoji jsem se mohl podílet na zajímavějších projektech této firmy a stát se plnohodnotným PHP vývojářem.

V rámci praxe jsem jako nováček dostal za úkol kódování šablony emailu pro hromadnou rozesílku. Na tvorbu šablony byly použity jazyky HTML a CSS, se kterými jsem se již setkal, ale své znalosti jsem zde mohl hojně rozšířit. V další fázi jsem byl přiřazen na samostatný projekt, kde mým úkolem bylo nakódovat šablony pro webovou stránku, připravit databázi MySQL a pomocí PHP frameworku Nette webovou stránku zprovoznit. Současně dalším úkolem bylo připravit program pro import dat o zájezdech cestovních kanceláří pomocí XML. V rámci tohoto úkolu byla důležitá rychlost zpracování a validita dat. Později jsem měl možnost vést tým a vyzkoušet si práci na jednom projektu ve více lidech, kde nestačilo umět programovat, ale bylo také důležité mezi sebou komunikovat a konzultovat další kroky a informovat ostatní členy týmu o případných změnách. V rámci praxe probíhaly také pravidelné porady jak interní v rámci firmy, tak porady přímo se zákazníkem, kde se projekt prezentoval.

Za dobu praxe jsem si prošel všemi pozicemi v IT oddělení. Od HTML/CSS kodéra, přes programátora webových aplikací až po PHP vývojáře aplikací zpracovávajících velké množství dat v řádu desítek gigabajtů a nakonec jsem skončil na pozici Vedoucího IT týmu. Díky tomu jsem si rozšířil znalosti HTML, CSS, JS, PHP a MySQL. Dále jsem se naučil pracovat s frameworkem Nette a nabral jsem spoustu zkušeností s organizací a vedením týmu.

2 Popis odborného zaměření firmy a pracovní zařazení

2.1 Odborné zaměření

Společnost INOvativní služby V IT s.r.o.[1] je českou firmou, která je na trhu druhým rokem. Zabývá se vytvářením řešení pro efektivní využívání informačních technologií. Vyvíjí a implementuje nástroje, které přizpůsobuje požadavkům klienta. Součástí činností firmy je dále propagace skrz PPC reklamy a její analýza.

Vlajkovým projektem je projekt Biddingtools[2], který zvyšuje obrat převáděním relevantních zákazníků a hlídá náklady, aby propagace přes zbožové porovnávače byla efektivní. Biddingtools funguje pro zbožové porovnávače Heureka.cz, Heureka.sk a Zboží.cz, s kterými je projekt také oficiálním partnerem.

Dále se firma věnuje vývoji webových projektů od nejmenších, jako jsou jednoduché webové stránky, přes složitější webové stránky a internetové obchody až k webovým stránkám pracující s velkým množství dat.

2.2 Popis pracovního zařazení

Po přijímacím pohovoru, jsem byl zařazen na místo kodéra webových stránek, kde jsem začal s tvorbou newsletteru a následně začal pracovat na tvorbě stránek v rámci odborné praxe. Po krátké době byla vyzkoušená má znalost PHP a dostal jsem se na pozici PHP programátora, kde jsem měl možnost se více seznámit s frameworkem Nette a s programovacím jazykem PHP.

Díky mým předchozím znalostem jsem byl také přeřazen do vývojového týmu projektu Biddingtools, kde jsem odbočil na čas od svého původního zadání odborné praxe. Nakonec jsem projekt po odchodu vedoucího pracovníka převzal a byl jsem přiřazen na pozici vedoucího IT týmu, kde nabírám zkušenosti s organizací, plánováním a vedením týmu.

[1] http://www.inovit.cz/
[2] http://www.biddingtools.cz/

3 Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti

3.1 Vytvoření webové prezentace Juhász

Zadání spočívalo v tvorbě webové stránky Juhász[3] sloužící jako online katalog zájezdů několika desítek cestovních kanceláří. Webová stránka měla za úkol poskytovat informace zákazníkům o zájezdech, ale také měla obsahovat interní část s rozšířeným výpisem pro prodejce. Kromě této možnosti měl mít zákazník možnost filtrovat zájezdy, dále si u zájezdu vybírat a filtrovat termíny a v neposlední řadě také si udělat kalkulaci zájezdu s možností nezávazné objednávky. Interní část pro prodejce měla rozšířenou i filtraci zájezdů o možnost filtrace dle cestovních kanceláří, čísla zájezdu a čísla termínu. Důležitým faktorem byla taktéž rychlost načítání webové stránky.

Práce	**Hodiny**
Hlavní stránka	20
Stránka s výpisem zájezdů	10
Detailní výpis zájezdu	15
Textová stránka	3
Programování funkčnosti webu	120
Programování filtrace zájezdů	40
Testování	20
Cachování a optimalizace webu	10
Celkem	**238**

Tabulka 1: Vytvoření webové prezentace Juhász - časová náročnost

[3] http://www.juhasz.cz/

3.2 Vývoj software pro získávání dat z XML o zájezdech CK

Zadáním úkolu bylo vytvořit software, který bude průběžně zpracovávat data z XML o cestovních zájezdech a jejich termínech a následně je ukládat do databáze. Aktualizace dat měla probíhat automaticky a neomezovat přitom webovou prezentaci. Program při spuštění měl za úkol vyhodnotit, které XML soubory jsou aktualizované od posledního zpracování a tyto soubory následně zpracovat a data uložit do databáze. Software měl dále udržovat aktuální tabulku s top zájezdy, a last minute zájezdy v případě, kdy nějaký termín prošel nebo byl již vyprodán. Požadavkem zadavatele byla možnost zpracovávat data některých cestovních kanceláří častěji než jen jedenkrát denně.

Práce	Hodiny
Celkem	**155**

Tabulka 2: Vývoj software pro získávání dat z XML o zájezdech CK - časová náročnost

3.3 Přepracování software pro zpracování XML dat o zájezdech CK

V tomto případě bylo úkolem přepracovat původní program na zpracování XML dat na nové XML z důvodu změny poskytovatele dat XML kvůli chybným datům o zájezdech cestovních kanceláří. Díky jinému přístupu nového poskytovatele se software měl rozšířit o pravidelnou hodinovou aktualizaci dat za pomoci rozdílových XML obsahující modifikovaná a smazaná data. Dalším krokem byla optimalizace programu a řešení kolizí na úrovni databáze při paralelním zpracování.

Práce	Hodiny
Celkem	**80**

Tabulka 3: Přepracování software pro zpracování XML dat o zájezdech CK - časová náročnost

4 Zvolený způsob řešení zadaných úkolů

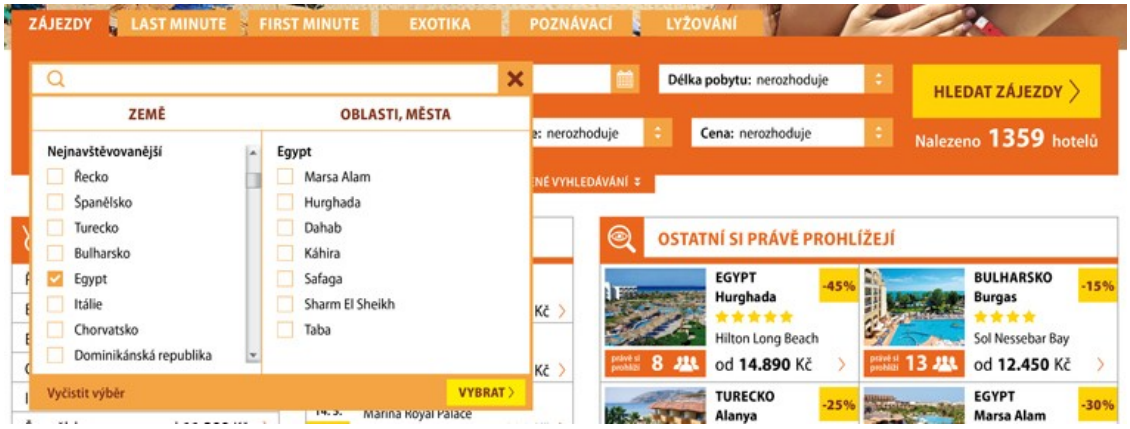
4.1 Vytvoření webové prezentace Juhász

Po zadání úkolu jsem dostal celkem čtyři varianty graficky navrhnutých šablon pro webovou prezentaci, kterou jsem měl připravit. Jednalo se o hlavní stránku, stránku s výpisem zájezdu, detailní stránku zájezdu a textová stránku zájezdu.

4.1.1 Stylování šablon

V rámci řešení úkolu jsem se rozhodl začít s kódováním šablon pro webové stránky. Kromě značkovacího jazyku HTML, stylovacího jazyku CSS a skriptovacího jazyku JavaScript jsem využil velmi populárního frameworku Bootstrap[4].Tento framework využívá jak HTML, CSS, tak i JS a díky několika předdefinovaných třídách zajišťuje jednoduché naimplementování responzivity webu. Responzivita webu v tomto případě nebyla důležitá. Počítalo se však, že web v budoucnu bude upravován pro lepší zobrazení na jak tabletech, tak i na telefonu. Vzhledem k tomu jak je projekt rozsáhlý a stylově náročný jsem v rámci optimalizace na základě kolegova doporučení použil Less[5]. Less je pre-procesor rozšiřující CSS o možnosti například proměnných a CSS kód dokáže převést téměř do kódu bez mezer a zbytečných znaků, a tak zajistit snížení velikosti rozsáhlých CSS souborů.

Začal jsem kódováním a stylováním hlavní stránky, která se ukázala být nejnáročnější. Kromě kódování hlavičky a patičky tu byl velmi složitý prvek filtr, který měl netypický design, jak můžete vidět na obrázku. Nedaly se zde použít klasické select boxy, a tak kromě stylování se společně se skriptovacím jazykem JavaScript musela celá funkcionalita filtru naprogramovat. Filtr dále obsahoval rozšiřující prvky jako kalendář, který se musel dostylovat dle návrhu a dále například našeptávač pro vyhledávání. Zbytek stránky nebyl tak náročný, obsahoval výpis top destinací, last minute zájezdy, zájezdů, které si ostatní prohlíží a top zájezdy. U top zájezdů se po najetí na zájezd měly zobrazovat rozšířené informace.



Obrázek 1: Formulář pro filtraci zájezdů

[4] http://www.getbootstrap.com/
[5] http://www.lesscss.org/

Dále jsem pokračoval tvorbou šablony s výpisem zájezdů, která se většinou zobrazovala po vyhledání zájezdů přes filtraci. Stránka měla obsahovat filtr jako na hlavní stránce. Bohužel struktura HTML filtru se musela změnit, aby zapadal do sloupce v levé části webu. Snažil jsem se zachovat jeden CSS styl filtru. Vzhledem ke složitosti designu filtrace jsem jeho styl rozdělil do samostatného CSS souboru. Webová šablona výpisové stránky vycházela z velmi podobných stylů jako hlavní stránka, a tak nebyla tolik náročná.

Následujícím stylovacím oříškem byla stránka s detailem zájezdu. Ta se rozkládala do několika částí. V první části byl název zájezdu, počet hvězdiček hotelu, lokalita a hlavně fotogalerie a tabulka se základními informacemi jako termín, délka pobytu, typ dopravy, strava a informace o ceně. Ve fotogalerii jsem musel řešit obrázky vysoké nebo naopak velmi široké. Jejich stylování jsem řešil pomocí vlastních CSS tříd *img_width* a *img_height*, které po načtení stránky přiděloval JS dle skutečných velikostí obrázku dle velikosti jeho stran. Toto se mi zdálo jako ideální cesta, jak rozlišovat vysoký nebo naopak široký obrázek. Tabulka s informacemi již tak těžká nebyla, ale opět stylově nebyla moc typická. Další část stránky se věnovala popisu zájezdu. Ten se skládal z nadpisu s obrázkem a daným popisem. Opět stylově ne moc typická část, ale naštěstí mi šla velmi jednoduše nastylovat. Třetí částí byla nezávazná objednávka prostřednictvím tabulky, poskytovala rychlou rekapitulaci zájezdu, kalkulaci zájezdu a po kliknutí na tlačítko nezávazně objednat se měl objevovat formulář pro vyplnění osobních údajů. Nejtěžší v této části bylo nastylovat kalkulaci zájezdu, která obsahovala opět specificky nastylovaný prvek formuláře, jak můžete vidět na obrázku. Sekci termíny a ceny stylovala kolegyně.

Obrázek 2: Formulář kalkulace zájezdu

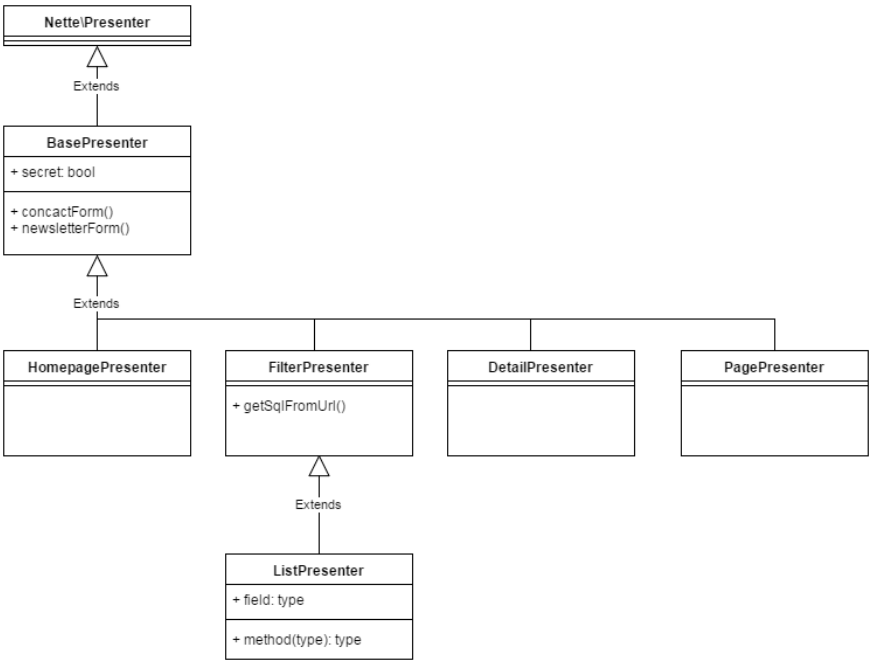
Poslední šablonou nutnou nakódování byla textová šablona. Ta byla nejlehčí, protože zde stačilo nakódovat jen odstavce a nadpisy. Další prvky byly použity z předchozích stránek.

4.1.2 Back-endová část webové stránky

Po vyřešení předchozích úkolů na řadu přišla back-endová část webové stránky. Ve firmě se na tvorbu webů používal PHP framework Nette[6], se kterým jsem měl minimální. Naskytla se mi tak příležitost, za podpory již zkušených programátorů naučit se více s daným frameworkem.

Protože jsem se frameworkem Nette teprve začínal, rozhodl jsem se s ním seznámit pomocí oficiální dokumentace a tutoriálu Píšeme první aplikaci[7]. Framework Nette je postavený na architektuře Model - View – Presenter[8], což je softwarová architektura, která vznikla z potřeby oddělit u aplikací s grafickým rozhraním kód obsluhy (Controller, u Nette je obdobou Presenter) od kódu aplikační logiky (Model) a od kódu zobrazujícího data (View) [1]. Pro zjednodušení může fungovat na architektuře Presenter - View, kde logika a dotazování přechází z modelu do presenteru.

Webové šablony byly hotové a na jejich základě a specifikace od klienta jsem od kolegů dostal třídní diagram rozdělení presenterů, který jasně definoval, jak budou části stránek odděleny. Z tohoto diagramu jsem vycházel. Při dalším návrhu a implementaci jsem se řídil analýzou cílů, která mi byla předána od firmy.



Obrázek 3: Třídní diagram rozdělení stránky do presenterů

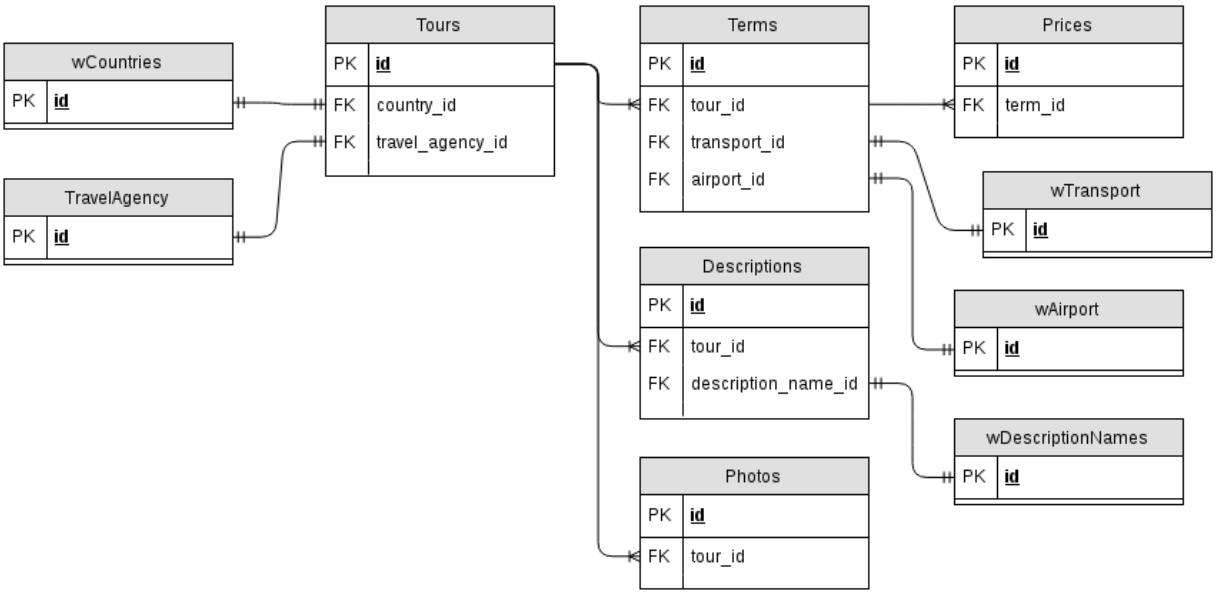
[6] http://www.nette.org/
[7] https://doc.nette.org/cs/2.3/quickstart/getting-started
[8] https://doc.nette.org/cs/2.3/presenters

Při tvorbě jsem se jako nezkušený řídil návodem pro začátečníky a oficiální dokumentací. Vytvořil jsem tak presentery pro hlavní stránku, stránku s výpisem zájezdů, stránku s detailem zájezdu a dále presenter pro textové stránky a pro stránku s oblíbenými zájezdy, která vycházela z designu výpisové stránky. Připravil jsem hlavní šablonu složenou s hlavičky a patičky. Dále jsem vytvořil obsahové šablony, které jsem rozdělil do složek pro šablony dle presenterů. Kvůli velké HTML náročnosti kódu jsem filtr na hlavní stránce a na výpisové stránce oddělil do zvláštních šablon pro lepší přehlednost kódu. Velkou chybou, kterou jsem jako začátečník učinil bylo, že jsem využíval balíčku Nette database [2], který byl určen ve spojitosti s Nette frameworkem pro Presenter - View architekturu, protože objekt databáze byl předávaný přímo do presenteru. Jakmile presentery začaly být složitější, musel jsem začít používat modely a objekt databáze do něj předávat skrz presentery. Toto bylo velmi nepraktické a kód se špatně udržoval. Výhodou bylo, že debagovací program Tracy [3], který je součástí Nette vyhodnocoval i rychlost SQL dotazů. Díky tomu se mohla první optimalizace stránky. Na tomto místě jsem se naučil používat lépe SQL dotazy a získávat data rychlejšími způsoby, protože na každé milisekundě záleží.

Webová prezentace byla téměř hotová, dalším krokem bylo zprovoznit filtrační formulář. Ten jsem navrhl tak, že data o filtraci se předávaly v URL, aby případná zaslaná URL adresa zobrazila stejný obsah jako uživateli, který ji zaslal. Filtr jsem tedy nastavil tak, aby při jakékoli změně přegeneroval URL, na kterou se bude dotazovat. Tyto parametry se při změně hodnot ve filtru rovnou zasílali pomocí AJAXu na stránku, která mu vracela počet termínů v databázi. Při odeslání formuláře byl uživatel přesměrován na stránku s výpisem zájezdů, kde se zpracovaly data z URL a přetvořily se do pole s podmínkami pro tabulku zájezdů a do pole s podmínkami pro tabulku termínu. Ty byly pak přidány do SQL dotazu. Tato část byla velmi náročná, protože se musely řešit různé druhy podmínek od rovnosti přes porovnávání menší, větší až po výběr prvků spadajících do množiny nebo také vyhledávání v textu.

Součástí takhle velkého projektu je samozřejmě testování. To se dělalo průběžně ve dvou až třech fázích. První fáze testování jsem se účastnil já a kolegyně, která se z menší části podílela na projektu se mnou, kdy jsme testovali věci, které se upravily. V případě, kdy kolegyně udělala nějakou práci, následovala fáze, kdy jsem po její kontrole práci kontroloval ještě já sám. Poslední fází byla kontrola lidmi zabývající se přímo projektem a kteří s ním budou pracovat. Díky tomu jsme měli zpětnou vazbu od slečen, které zájezdy prodávají a dokázaly nás upozornit na chyby, které jsme přehlédli, nebo z důvodu neznalosti za chyby ani nepovažovali.

Konečnou fází byla optimalizace webu. V této fázi jsem optimalizoval SQL dotazy a také kód. Pomocí Tracy – debugovacího programu Nette, jsem sledoval, které SQL dotazy se volají při načítání stránky a jak dlouho trvá jejich vykonání. Díky této analýze jsem zjistil, že nejdéle se vykonávají dotazy JOIN nad závislými tabulkami. Řešení jsem nalezl v článku od pana Jakuba Vrány, kde srovnával dotazy do závislých tabulek [1]. Podle tohoto článku jsou tři možnosti dotazu do závislých tabulek. První možností je položit jeden komplexní dotaz, což jsme v našem případě prováděli. Další možností je položit konstantní počet dotazů, pro každou tabulku jeden. Třetí možností bylo položit více dotazů, například jeden dotaz pro každý záznam v hlavní tabulce. Původně jsem se řídil pravidlem, že by počet dotazů do tabulky měl být co nejnižší. Kromě minimalizace počtu dotazů jde ale při komunikaci s databází i o to, kolik dat se přenáší. V rámci jednoho komplexního dotazu se znovu přenáší data, které už máme. To znamená, že pro každý řádek ze závislé tabulky se znovu přenáší i data z hlavní tabulky. U druhého řešení, kdy posíláme na databázi konstantní počet dotazů se přenášejí všechna data kromě primárních klíčů jen jednou. V praxi to tedy znamená, že z databáze získáváme menší objem dat než při komplexním dotazu a tedy celý chod aplikace je rychlejší. Rozdělil jsem tedy SQL dotazy do částí a propojení dat jsem řešil až na úrovni programu. Dosáhl jsem toho tak, že například zájezd a termíny byly tabulky 1:N a tedy tabulka termínu obsahovala cizí klíč na tabulku zájezdů. Když si například vezmeme, že potřebujeme získat data pro detail stránky, kde známe id zájezdu a id termínu, tak jsem data původně získával za pomocí jednoho dotazu pomocí JOIN. V rámci optimalizace jsem dotazy rozdělil na dva. Jeden vytáhl data z tabulky pro zájezdy a druhý dotaz vytáhl data z tabulky termínů. Na úrovni programu pak bylo jednoduché data z tabulek spojit. Díky této optimalizaci se mi podařilo webovou stránku zrychlit. Na obrázku můžete vidět zjednodušený databázový model znázorňující propojení tabulek v rámci databáze za pomocí cizích klíčů.



Obrázek 4: Zjednodušený datový model

Po optimalizaci dotazování do MySQL databáze jsem začal řešit ladění kódu. Jak jsem výše zmínil, předávání objektu databáze skrz presenter do databáze bylo velmi nepraktické. Z toho důvodu jsme začali používat databázovou knihovnu dibi[9], která je statická a tedy jsme měli globální přístup k dostupnému uložišti objektu spojení a nad ním jsme mohli volat potřebné metody. S tímto přepisem mi paralelně pomáhala kolegyně. Další fází optimalizace bylo cachování obsahu, tedy uložení dat z databáze na určitou dobu do souboru. K tomu jsem využil PHP třídu Cache, která již byla součástí frameworku Nette. V první fázi jsem uložil do cache dynamické části hlavní stránky, jako byly last minute zájezdy, top zájezdy nebo zájezdy, na které se zrovna někdo dívá. Cache jsem ukládal na dobu 30 až 40 minut, a tak docílil toho, že ⅔ webu vždy naběhnou rychle a jen ⅓ se nahraje z databáze. Dalším krokem bylo natáhnout si všechny číselníky z databáze. Tímto krokem jsme zamezili opakovanému dotazování do tabulky se statickými a hodnotu jsme získali přímo z pole, kde klíčem bylo id a hodnotou hodnot sloupce name. Následně jsem se zaměřil na cachování ostatních SQL dotazů, a tak zájezdy a termíny jsem začal cachovat. Tuto cache jsem ukládal na dobu dvou hodin a tím jsem dosáhl toho, že nejnavštěvovanější zájezdy vždy naběhli rychle a v případě změny se minimálně co dvě hodiny znovu natáhli data z MySQL databáze, původní cache se smazala a nahradila se cache s aktuálními daty, která opět byla uložena na dobu dvou hodin.

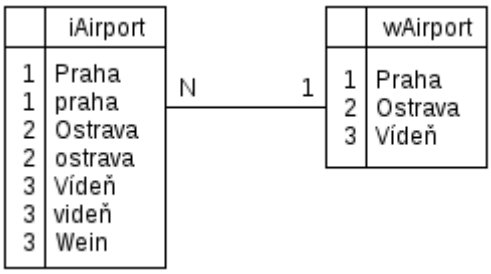
[9] http://www.dibiphp.com/

4.2 Vývoj software pro získávání dat z XML o zájezdech CK

Na tomto úkolu jsem pracoval paralelně s tvorbou webové prezentace. Na začátku jsem dostal odkaz na XML třetí strany s daty o zájezdech různých cestovních kanceláří. Úkolem bylo navrhnout databázi a data zpracovávat a unifikovat.

Program ze začátku vypadal velmi jednoduše. Program měl se dotázat na dané URL s XML souborem, který obsahoval seznam XML souborů podle cestovních kanceláří a datum poslední aktualizace. V XML souborech cestovních kanceláří byly data logicky uspořádaná, a tak jsem neviděl žádný problém ve vytvoření tohoto programu.

Prvním krokem bylo vytvořit tabulky. Tabulky jsem vytvořil pro zájezdy (Tours), popisy zájezdu (Descriptions), termíny zájezdu (Terms), fotografie zájezdu (Photos) a ceny k danému termínu (Prices). V rámci unifikace jsem vytvořil několik dalších tabulek - číselníky. Vždy jeden číselník, ze kterého četl import s prefixem “i”, který se skládal ze sloupců id a name. Unikátním klíčem bylo name a id mohlo být duplicitní. Dále číselník, ze kterého četl web s prefixem “w”, který se skládal také z id a name. Na tomto místě bylo tentokrát id klasicky unikátní. Tyto číselníky byly například pro letiště, stravování, země, lokality, názvy popisů a tak podobně. Ukázku číselníku pro letiště můžete vidět na obrázku. Později bude jasné, proč jsem takhle databázi navrhl. Stejně tak jsem navrhl i tabulku pro cestovní kanceláře, která pro import měla rozšíření o sloupec s datem poslední aktualizace. Poté jsem vytvořil tabulku logNotLoaded pro záznamy, které nebyly v číselníku.



Obrázek 5: Ukázka číselníku pro letiště

Dalším logickým krokem bylo vytvoření programu. Program jsem nevytvářel s pomocí frameworku, ale za pomoci balíčků PHP souborů, ze kterých jsem projekt seskládal dohromady. Balíčkem, který jsem použil byla třeba Tracy od Nette, kterou jsem použil jako debugger. Původně, stejně jako při tvorbě webové stránky, jsem použil balíček Nette database. Ta mě přinutila najít něco víc praktického, takže jsem začal používat balíček dibi jako databázovou vrstvu. Posléze jsem je také využil ve webové části.

Program pracoval tak, že si z databáze vytáhl data poslední aktualizace. Poté se dotázal na XML soubor se seznamem XML souborů cestovních kanceláří a daty jejich poslední aktualizace. Soubory, které měly datum aktualizace novější, než bylo v databázi, byly staženy a začaly se zpracovávat. Pro zpracování jsem použil SimpleXML, jednu ze základních tříd PHP pro zpracování XML. Při tvorbě programu jsem velmi využíval oficiální dokumentaci PHP [5]. Než se XML soubory začaly zpracovávat, natáhl jsem si do pole všechny data z číselníkových tabulek označené prefixem "i" a klíčem pole bylo name a hodnotou id. Při projíždění uzly XML, když jsem narazil například na název země, která byla uvedena textově. Už kvůli rychlejšímu vyhledání jsem potřeboval text převést na id. Vytvořil jsem tedy metodu, do které jsem poslal název země, ta vzala pole s číselníkem země, použila název země jako klíč pole a vrátila tak hodnotu id. Pokud klíč v poli neexistoval, znamenalo to, že tuto zemi neznáme. Název země byl tedy uložen do tabulky logNotLoaded, ze které po importu byl ručně předáván do svého číselníku a bylo mu přiděleno i id. Díky tomu, že id v číselnících pro import mohlo být duplicitní, dosáhl jsem toho, že funkce při dotazu například na "Spojené Arabské Emiráty" a "SAE", vrátila vždy stejnou hodnotu id. V praxi, když se web dotázal do číselníku s prefixem "w", měl již jen jedno id a pod ním měl unifikovaný název. Vždy při dokončení průchodu uzlem XML, jsem volal SQL pro vložení do databáze. Což celkem dobře fungovalo na malém počtu dat, na kterém jsem projekt testoval.

Když jsem začal projekt testovat na větším počtu dat, zjistil jsem, že jednotlivé SQL dotazy trvají dlouho a celý program by běžel několik hodin, možná i desítek hodin nebo dnů. Při hledání na internetu jsem zjistil, že nejrychlejší je vkládat do MySQL databáze za použití CSV souborů. Poté jsem si tuto informaci ověřil dle oficiální dokumentace MySQL, kde uvádí, že vkládání do databáze z textového souboru je dvacetkrát rychlejší, než obyčejný INSERT [6]. Předělal jsem tedy projekt tak, že výsledky z jednotlivých XML uzlů jsem si ukládal do pole, které jsem po zpracování XML uložil do CSV souboru. Tento CSV soubor se poté importoval do MySQL databáze. Přes CSV soubory jsem vykonával pouze operaci INSERT nad databází. Mazání starých záznamů z databáze jsem vykonával přes SQL dotaz DELETE. SQL dotaz mazal data z MySQL databáze podle množiny primárních klíčů. Pokud tedy primární klíč náležel do množiny, byl smazán. Tento SQL dotaz se díky vyhledání a mazání podle primárních klíčů vykonával velmi rychle. MySQL databáze samozřejmě podporovala Multi INSERT, ten byl bohužel omezen délkou SQL dotazu, navíc import CSV souborů je stále rychlejší metodou pro operaci INSERT. To jsem si ověřil při otestování na několika cestovních kancelářích. Zjistil jsem, že se zpracování XML zrychlilo za požití importu skrz CSV soubor do MySQL databáze oproti klasické operaci INSERT. Následně jsem začal testovat zpracování všech XML cestovních kanceláři. Po několika hodinách práce však se naplnily paměti RAM a program byl sestřelen systémem. Na radu zkušenějšího kolegy, jsem převedl program na výkonnější server. Bohužel po pár hodinách běhu program opět zahltil z neznámých příčin paměť RAM. Nyní však systém neměl žádnou ochranu a přetížil se natolik, že došlo k restartu celého serveru, kde program běžel. Se zkušenějším kolegou jsme tedy začali hledat chybu. Zakomentovávali jsme různé části kódu, až jsme došli téměř k prvnímu řádku. Zjistili jsme, že PHP třída SimpleXML má v sobě chybu, která si v paměti RAM drží stále data až do ukončení PHP skriptu. Konkrétně chyba nastává při načítání XML souboru. Tento problém jsem musel nějak vyřešit. Abych nemusel celý program předělávat, hledal jsem řešení jak chybu obejít. Po několika hodinách hledání a zkoušení různých metod jsem přišel s řešením. XML soubor jsem začal načítat za pomocí PHP třídy XMLReader a DOMDocument, díky kterým jsem načtený XML soubor mohl převést do PHP třídy SimpleXMLElement a celý program tak mohl pracovat dál bez většího zásahu do celého kódu.

Webová stránka byla hotova, program připraven. Bohužel všechny cestovní kanceláře program dokázal zpracovat přibližně za 8 hodin. Kvůli tak dlouhé době zpracování nebylo možné data cestovních kanceláří zpracovávat několikrát denně. Program byl vyladěn a další možnosti jak program optimalizovat a zrychlit mě nenapadali. Omezením programu byl tedy hardware. Musel jsem program přesunout a data začít zpracovávat na jiném serveru, který není zatěžován jinými programy. Firma v rámci vývoje svého nástroje Biddingtools se zabývala moderní technologií Google Cloud služeb [7]. Ty poskytují možnost pronájmu virtuálních strojů, tak zvaných Compute Engine, dále Cloud SQL fungující jako běžná MySQL databáze nebo služby App Engine s možností přístupu přes HTTP a HTTPS protokol a případným spuštěním za pomoci CRONu. Hlavním omezením App Enginu bylo, že při zavolání požadavku musel program vrátit odpověď nejpozději do šedesáti sekund, jinak byl program předčasně ukončen. Další nabízenou službou byl Google Bucket, který se využívá jako uložiště. Tyto služby jsou zpoplatněny dle toho, jak jsou časově využívány. Tedy za virtuální stroj platíte pouze když běží. Stejně tak za App Engine a databáze. U databází a Bucketu platíte navíc dle velikosti uložených dat. Celý tento systém je postavený na API a tak můžete například vypínat a zapínat virtuální stroje když je potřebujete za pomoci programu.

Možnost předělat program na Google Cloud služby byla velmi lákavá, protože data by se mohli zpracovávat na vlastním serveru za relativně málo peněz. Připravil jsem tedy jeden virtuální stroj, který měl zpracovávat data. Na našem serveru jsem připravil MySQL databázi do které se data měli zapisovat a povolil jsem přístup pro IP adresu serveru Google. Při prvním testování nastal problém, protože nefungoval vzdálený import souboru CSV do MySQL databáze na našem serveru. Původním nápadem bylo nainstalovat MySQL databázi na serveru Google. Import CSV souborů do databáze na stejném serveru sice fungoval, ale server by musel běžet 24 hodin denně, aby byla databáze stále dostupná pro webovou stránku. Toto řešení rázem nebylo finančně zajímavé. Vytvořil jsem tedy Cloud SQL, MySQL databázi, která když není používaná, tak se její instance se vypne. Jakmile je na ni vykonán požadavek znovu se zapne. Nastavil jsem tedy veškerá oprávnění, aby server mohl komunikovat s Cloud SQL. Při dalším spuštění jsem se opět zdržel na problému s importem CSV souborů do databáze. Takový import do Cloud SQL nelze udělat, pokud CSV soubor není v Google Bucketu a požadavek na import musí být proveden za pomocí API. Přišla tedy další úprava programu, kde jsem přidělal nahrávání CSV souborů na Bucket. Kvůli velikosti museli být soubory posílány po částech a nikoliv jako celý soubor na API Bucketu. Tuto funkčnost v obecném kódu definovalo dokumentace Google. Při této úpravě programu jsem taky přidal možnost volání Cloud SQL API pro import souborů. Cloud SQL nedovolovala import několika souborů a tak jsem se musel dotazovat na API, jestli předchozí soubor už byl importován do databáze. Kvůli omezenému počtu dotazů na API jsem rozestupy dotazů na API, zda soubor byl importován do databáze určoval dle velikosti souboru. Tedy jestliže soubor byl velký, rozestupy dotazů byly několik sekund. Pokud naopak CSV soubor byl malý, tak se program na API dotazoval po sekundě. Ve většině případů byl soubor již importován, nebo proběhli dva až tři dotazy na API než import doběhl do konce.

Dostal jsem se do stavu funkčního programu na samostatném serveru. Bohužel způsob importu značně zpomalil celý proces zpracování. Napadlo mě import oddělit od programu a paralelně data importovat. Na toto jsem využil App Engine. Vytvořil jsem v něm jednoduchý skript, který procházel složky na Google Bucketu pomocí API a následně pomocí Cloud SQL API vysílal požadavek na import CSV souborů do databáze. Poté se dotazoval na API, zda soubor byl již importován a následně jej smazal z Google Bucketu. Krok to byl správný. Zpracování se zrychlilo a data se průběžně importovali do databáze. Začalo se však stávat, že z důvodu již existujícího primárního klíče se podařilo importovat pouze část souboru a spousta dat chyběla. To jsem vyřešil pomocí souboru *.lock*, který jsem vytvořil ve složce Google Bucketu pro danou cestovní kancelář. Ten zajišťoval, že když složku procházel App Engine za účelem importu, tak první soubor který přečetl, byl právě *.lock* soubor. Díky tomu App Engine složku opustil a nic neimportoval. Jakmile program zpracovávající data dané cestovní kanceláři zpracoval, promazal databázi a smazal *.lock* soubor. Při dalším průchodu složky App Engine začal data importovat do databáze, pokud *.lock* soubor ve složce již nebyl. Z důvodu omezených počtu dotazů na API všech služeb od Google jsem musel upravit program na větší rozestupy pro dotazování. Několikrát se stalo, že program tohoto omezení dosáhl a má práce se na tomto projektu zastavila do dalšího dne, než se limity vynulovali.

Toto předělání zrychlilo program a data se začali zpracovávat místo původních 8 hodin za přibližně 6 hodin. Kvůli omezení počtu dotazu na API v základním režimu použití se data dali zpracovávat opět pouze jedenkrát denně. Naštěstí v rozšířené placené verze se limity dali navýšit, a tedy zde byla možnost data v budoucnu zpracovávat pravidelněji několikrát denně. Těšil jsem se na první spuštění. Bohužel v programu bylo několik chyb, které byly důvodem špatně vkládaných dat do databáze. Po vyladění mohl jít projekt do světa. Má práce nekončila a dále jsem komunikoval se zákazníkem a vylepšoval XML import a celý projekt. Hlavní problém byl, že program ke spoustě zájezdům nenačítají fotky. Tento problém jsem se snažil vyřešit, ale mou odpovědí bylo, že fotografie k daným zájezdům v XML souboru nejsou. Dalším a zásadním problémem bylo, že u některých cestovních kanceláří nesedí ceny, respektive ceny jsou nižší než má daná cestovní kancelář v oficiální nabídce. Tento problém jsem řešil delší dobu. Došel jsem k tomu, že tyto ceny jsou opravdu v XML souboru. Přemýšlel jsem, jestli třeba nejsou ceny u některých CK zasílány bez DPH, ale ani tak to nevycházelo. Kontaktoval jsem tedy poskytovatelem XML, se kterým jsme problém řešili. Po několika dnech telefonátů a emailové komunikace vyšlo najevo, že chybovosti jsou si vědomi a není v jejich silách to opravit. Rozhodli jsme se přejít tedy k jinému poskytovateli dat o cestovních zájezdech a tak mohlo téměř vše začít od začátku.

4.3 Přepracování software pro zpracování XML dat o zájezdech CK

Úkol byl nyní zcela jasný. Měl jsem přepracovat původní program na nová data, upravit databázi a celou webovou prezentaci. Nespornou výhodou bylo, že nový poskytovatel dodával svoje číselníky, a tak jsme nemuseli tvořit vlastní číselníky od nuly. Strukturu databáze jsem se snažil zachovat. Jednalo se o podobná, ale mnohem kvalitnější data. Navíc jsem nemusel tolik zasahovat do webové prezentace kvůli změnám.

Program jsem se rozhodl optimalizovat, a tak místo čtení pomocí PHP tříd XMLReader, DOMDocument a zpracování skrz SimpleXMLElement, jsem použil PHP třídu SAX, která je obecně doporučována pro rychlé čtení XML. SAX není tolik uživatelsky přívětivá PHP třída, jako SimpleXML, ale opravdu zrychlila čtení XML. Pozměnil jsem také adresářovou strukturu a udělal jsem program tak mnohem přehlednější. Import pomocí CSV souborů jsem ponechal, pouze jsem použil PHP třídu SplFileObject pro zápis kvůli validitě dat. Velkým krokem bylo paralelní zpracování souborů pomocí procesů, které jsem povýšil navíc na asynchronní zpracování, jak můžete vidět v ukázce zdrojového kódu.

```php
$fork = new Fork();
$pids = [];
foreach ($travel_agency_ids as $id) {

    $conn = dibi::getConnection();
    try {
        /*
         * Vytvoří nový proces a vrátí jeho PID
         */
        $pid = $fork->call(function () use (&$parser, $id, $conn) {

            $start = microtime(true);
            DataManager::setCsvDir(CSV_DIR."{$id}/");
            $conn->connect();
            $parser->parseData(URL_MASTERS . "?to={$id}", $id);
            $time = Time();
            $parser->parseData(URL_DATA . "?to={$id}", $id);
            $stop = microtime(true);
            Debugger::log("Parse data time {$id}: " . ($stop - $start), Debugger::INFO);

            $i_start = microtime(true);
            DataManager::importAllCsv();
            $i_stop = microtime(true);
            Debugger::log("Import time {$id}: " . ($i_stop - $i_start), Debugger::INFO);

            $parser->deleteOldRows();

            DataManager::updateBigUpdate($id, $time);

        });

        // Přidá PID do pole aktivních procesů
        $pids[$id] = $pid;

    } catch (Exception $e){
        Debugger::log("E - {$id}: ".$e->getMessage(), Debugger::ERROR);
    }

    // Počet jader serveru/Počet paralerních zpracování
    $cores = 5;
    $count = count($pids)%$cores;
    /*
     * Pokud dosáhne maximálního počtu paralerně spuštěných procesů,
     * kontroluje je každou sekundu zda běží.
     * Pokud je aktivních procesů méně než maximum while se nevykoná
     * a pokračuje se ve spouštění dalších procesů
     */
    while($count == 0) {

        foreach($pids as $id => $pid) {
            $output = "";
            // zjištění zda proces běží
            exec("ps -p {$pid}", $output);
            //Proces doběl, mažeme jeho PID z pole aktivně běžících procesů
            if(strpos($output[1], "defunct"))
                unset($pids[$id]);
        }
        sleep(1);
        $count = count($pids)%$cores;
    }

}

// čekáme až doběhne zbytek procesů
$fork->wait();
```

Výpis 1: Asynchronní zpracování pomocí procesů

Poskytovány byly také XML soubory, které byly aktualizovány každou hodinu s daty o aktualizaci některého z termínů nebo o jeho smazání. Program jsem tedy rozšířil a XML data jsem začal také paralelně zpracovávat. Výhodu paralelního zpracování bylo využití všech jader procesoru a dosáhnutí tak rychlejšího zpracování. Velkou nevýhodu pak byly kolize v databázi, kdy spousta SQL dotazů se nevykonala z důvodu zamknuté tabulky a následného pádu dotazu, nebo se také často stávalo, že se databáze odpojila a přestala komunikovat. Toto jsem vyřešil vlastní třídou v PHP, která dědila po třídě dibi a tyto výjimky ošetřovala. Podívat se na ni můžete v ukázce zdrojového kódu

```php
/**
 * Generates and executes SQL query - Monostate for Dibi\Connection::query().
 * @param  array|mixed      one or more arguments
 * @return Dibi\Result|int  result set object (if any)
 * @throws Dibi\Exception
 */
public static function query($args)
{
    $args = func_get_args();

    try{
        return self::getConnection()->query($args);
    }catch(DriverException $e){

        switch(intval($e->getCode())){

            //MySQL server has gone away
            case 2006:
                Debugger::log("MySQL 2006: reconnect", Debugger::ERROR);
                self::reconnect();
                break;
            //Deadlock found when trying to get lock; try restarting transaction
            case 1213:
                Debugger::log("MySQL 1213: restarting transaction", Debugger::ERROR);
                sleep(1);
                break;
            //Lock wait timeout exceeded; try restarting transaction
            case 1205:
                Debugger::log("MySQL 1205: Lock wait timeout", Debugger::ERROR);
                sleep(60);
                break;
            default:
                Debugger::log($e->getMessage(), Debugger::ERROR);
                throw $e;
                break;

        }

        return self::query($args);

    }

}
```

Výpis 2: Třída MyDibi

Program byl tedy hotový. Relativně rychle se jej podařilo odladit a tak byla připravena databáze a naplněna validními daty. Data se navíc nyní zpracovávali přibližně za 4 hodiny a poté se každou hodinu pravidelně aktualizovala z rozdílových XML souborů, které jsme od poskytovatele dat získávali. Následovalo pár úprav back-endové části webové stránky a projekt mohl být zveřejněn.

5 Teoretické a praktické znalosti a dovednosti získané v průběhu studia a uplatněné studentem v průběhu odborné praxe

V průběhu praxe jsem využil zejména teoretické znalosti získané během studia, které jsem mohl prakticky aplikovat. Zejména se mi přišly vhod návrhové vzory, které jsem se učil v předmětu Úvod do softwarového inženýrství. Díky těmto znalostem mi nedělalo problém pochopit stavbu frameworku Nette, který je postavený na MVP architektuře a dále ani jeho komponent, například továrničky, která se používá pro tvorbu formulářů. Dále jsem využil znalostí z předmětu Úvod do databázových systémů, který mě naučil i složitější dotazy nad SQL tabulkami, které jsem využil pří práci s MySQL databází. Dále bych neměl opomenout ani předmět Uživatelské rozhrání, díky kterému jsem mohl přispět k návrhu webu a rozmístění určitých komponent v šabloně, aby byly uživatelsky přívětivější. Většinu nových znalostí jsem nabíral skrz dokumentace, které byly napsány v angličtině, a proto jsem uplatnil i znalosti z předmětu Anglický jazyk, kde jsem se naučil slovíčka z odborné angličtiny, které jsem uplatnil.

6 Znalosti a dovednosti scházející studentovi v průběhu odborné praxe

V průběhu praxe mi rozhodně scházela hlubší znalost jazyku PHP, který se na škole nevyučuje, přestože v Ostravě je velká poptávka po PHP programátorech. Avšak tento nedostatek mi nahrazovala znalost jiných programovacích jazyků, ze kterých jsem mohl vycházet. Dále mi chyběla znalost optimalizace programu, kde se pracuje s větším počtem dat. Ve škole jsme dělali jednoduché aplikace s databázemi obsahující pár řádků, kde optimalizace nebyla zapotřebí. V praxi jsem pracoval běžně s databázemi o stovkách tisíc až milionech řádků. Později v průběhu mé praxe jsem se dostal na pozici vedoucího IT týmu, kde mi chyběla znalost ve vedení a organizace týmu. Naštěstí tyto znalosti a dovednosti mě byla firma schopna postupně naučit.

7 Dosažené výsledky v průběhu odborné praxe a její zhodnocení

Díky této praxi jsem měl možnost rozšířit své znalosti. Zdokonalil jsem se v HTML, CSS i JavaScriptu. Naučil jsem se pracovat s frameworkem Bootstrap. Seznámil jsem se s rozšířením CSS jazyka Less, dále jsem posunul své znalosti PHP na vyšší úroveň a naučil jsem se pracovat s frameworkem Nette a dalšími balíčky PHP. Navíc jsem získal cenné zkušenosti v oblasti organizace, plánování a vedení týmu. Oproti normální bakalářské práci jsem měl možnost se reálně zapojit do projektů, které fungují, mohl jsem spolupracovat v týmu a sledovat jak projekt funguje. Tyto získané zkušenosti se mi rozhodně budou hodit v profesním životě.

S praxí ve firmě jsem nad míru spokojen a jsem rád, že jsem si vybral právě INOvativní služby V IT s.r.o., která mě postavila před různé výzvy, kterým jsem se mohl postavit a dokázat si, že je zvládnu. Navíc mi poskytla skvělé pracovní zázemí v prostorách IMPACT HUB Ostrava a úžasný kolektiv lidí díky kterým jsem získal spoustu nových znalostí.

8 Reference

[1] **Nette Foundation.** MVC aplikace & presentery. *Nette framework.* [Online] © 2008-2016. [Citace: 4. duben 2016.] https://doc.nette.org/cs/2.3/presenters.

[2] **Nette Foundation.** Databáze. *Nette framework.* [Online] © 2008-2016. [Citace: 1. duben 2016.] https://doc.nette.org/cs/2.3/database.

[3] **Nette Foundation**. Debugování a zpracování chyb. *Nette framework.* [Online] © 2008-2016. [Citace: 1. duben 2016.] https://tracy.nette.org/cs/.

[4] **Vrána, Jakub.** Srovnání dotazů do závislých tabulek. *PHP triky.* [Online] © 2005-2016. [Citace: 1. duben 2016.] http://php.vrana.cz/srovnani-dotazu-do-zavislych-tabulek.php.

[5] **PHP Documentation Group.** PHP Manual. *PHP: Hypertext Preprocessor.* [Online] © 1997-2016. [Citace: 1. duben 2016.] http://php.net/manual/en/.

[6] **Oracle Corporation and/or its affiliates.** Speed of INSERT Statements. *MySQL.* [Online] © 2016. [Citace: 1. duben 2016.] http://dev.mysql.com/doc/refman/5.7/en/insert-speed.html.

[7] **Google Ireland Limited.** Products & Services. *Google Cloud Platform.* [Online] © 2016. [Citace: 1. duben 2016.] https://cloud.google.com/products/.